The Kepler end-to-end model: creating high-fidelity simulations to test Kepler ground processing

Stephen T. Bryson[a], Jon M. Jenkins[b], Dan J. Peters[c], Peter P. Tenenbaum[b] Todd C. Klaus[d], Jay P. Gunter[d], Miles T. Cote[a], Douglas A. Caldwell[b],

[a]NASA Ames Research Center, M/S 244-30, Moffett Field, CA USA 94305
[b]SETI Institute/NASA Ames Research Center, M/S 244-30, Moffett Field, CA USA 94305
[c]Ball Aerospace & Technologies Corporation, 1600 Commerce Street, Boulder, CO USA 80301
[d]Orbital Sciences Corporation/NASA Ames Research Center, M/S 244-30, Moffett Field, CA USA 94305

ABSTRACT

The *Kepler* mission is designed to detect the transit of Earth-like planets around Sun-like stars by observing 100,000 stellar targets. Developing and testing the *Kepler* ground-segment processing system, in particular the data analysis pipeline, requires high-fidelity simulated data. This simulated data is provided by the *Kepler* End-to-End Model (ETEM). ETEM simulates the astrophysics of planetary transits and other phenomena, properties of the *Kepler* spacecraft and the format of the downlinked data. Major challenges addressed by ETEM include the rapid production of large amounts of simulated data, extensibility and maintainability.

Keywords: *Kepler*, pixel simulation

1. INTRODUCTION

The *Kepler Mission* continuously observes ∼165,000 target stars in *Kepler's* 115 square degree Field of View (FOV) seeking to discover Earth-like planets transiting solar-like stars by detecting photometric signatures of transits.[1,2] Data is collected and stored for monthly downlink, and the data is processed in the Science Operations Center (SOC).[3,4]

In order to test the *Kepler* ground system the *Kepler* End-to-End Model (ETEM) was developed which provides test data formatted as it appears when received by ground tracking stations and contains astrophysically realistic.

Kepler operations has three phases: data collection, monthly data downlink and data processing to identify transits and other astrophysical phenomena. These phases involve several organizations that must interface smoothly. ETEM is designed to simulate the collected data, including high-fidelity simulations of astrophysical processes including transit signals and stellar variability, as well as spacecraft noise and systematics. ETEM packages this data in a way that mimics the data as it appears to NASA's deep space network. Therefore a single ETEM-generated data set can be used to test every step of *Kepler* data flow and processing, starting with the arrival of the data on the ground and ending with the identification of planetary transits.

To support testing and development of *Kepler* operations ETEM is designed as a robust, extensible system that allows the easy addition of new phenomena to be simulated. This paper is a description of the addition of extensibility and robustness to ETEM beyond the system described elsewhere.[13] After describing the *Kepler* system and data simulated by ETEM, we describe the astrophysical simulation methods (§2) and the software class and plugin structure (§4). We end the paper with a brief discussion of the use of ETEM data in testing (§5).

Further author information: Send correspondence to S.T.B.: E-mail: Steve.Bryson@nasa.gov

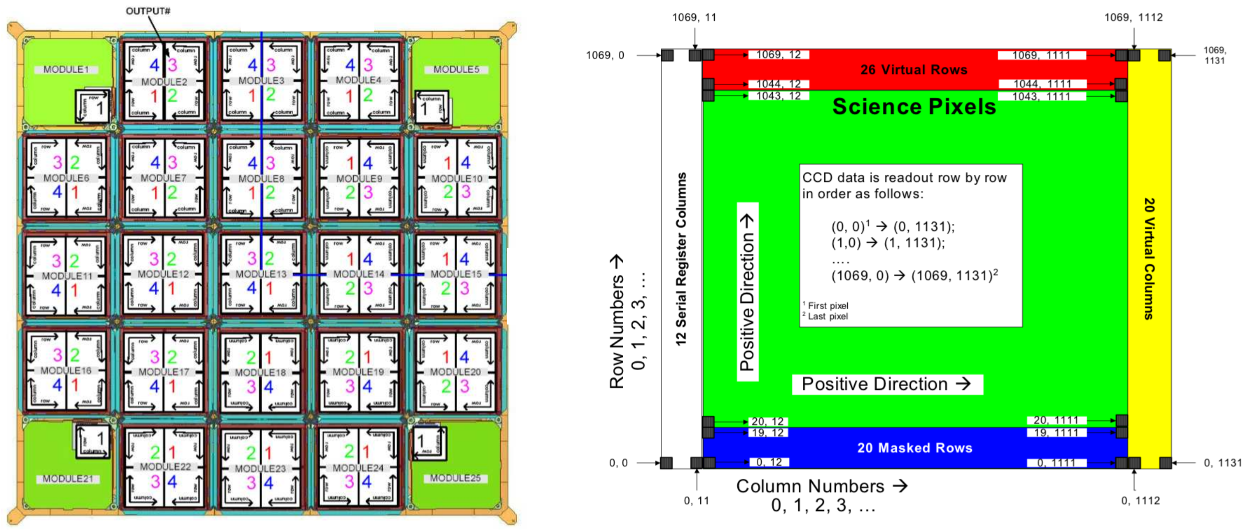


Figure 1. Left: The CCD array on the *Kepler* focal plane, showing the 21 modules, each of which has 2 2200 x 2048 pixel CCDs. Each CCD is read out via 2 output channels. Smaller CCDs used as fine guidance sensors are also shown in each corner, but are not discussed in this paper. Right: the pixel arrangement of each output channel.

1.1 The *Kepler* focal plane

The *Kepler* focal plane science sensors consist of 42 2200 column by 1044 row CCDs mounted on 21 electronic modules (Fig 1) with an image scale of 3.98″per pixel.[5,6] The first 20 rows of each CCD are not exposed to the sky in order to provide calibration and diagnostic data as described below. Each CCD is divided into two 1112×1044 *output channels*. Each output channels is supplemented by 26 trailing virtual rows, 12 leading serial register columns and 20 trailing virtual columns, giving each output channel 1136×1070 addressable pixels. The 12 leading serial register columns and 20 trailing virtual columns are used to collect black level data for each row. *Kepler's* lack of a shutter means that pixels are exposed to the sky during readout, which causes image smear along columns. The leading 20 masked and 26 trailing virtual columns measure this smear data. The black level and smear data are called *collateral data* and are used to calibrate the pixel data during ground processing.[7] The pair of CCDs on each module provide a contiguous 2200×2048 pixel image of a portion of the *Kepler* field of view.

Kepler is designed to continuously observe one field of the sky with fixed pointing throughout the mission. Nonetheless there are two types of motion that must be simulated: residual pointing jitter and differential velocity aberration (DVA) due to the very wide *Kepler* field of view. DVA can move a star as much as 0.6 pixels in a quarter.

Every *Kepler* target is observed with a CCD readout every 6.52 seconds and co-added into 29.4 minute observations, referred to as *Long Cadence* (LC) data. A smaller number of targets, at most 512, is co-added into 58.8 second *Short Cadence* (SC) observations. These data are collected nearly continuously for about 30 days and downlinked via high-bandwidth Ka-band transmissions. Bandwidth constraints of the Ka-band transmission limits the amount data that may be downlinked, making it impossible to downlink all 96 million pixel values that are collected with each LC observation. Each target is therefore assigned a set of pixels sufficient to recover the flux of that target,[8] which provide the pixel flux time series in either short or long cadence. In addition all pixels are occasionally downloaded, nominally once a month, producing a *full frame image* (FFI).

1.2 *Kepler* pixel and target types

Kepler pixels are collected for several types of targets:

Stellar targets[9] are point-like sources whose pixels are selected to maximize the signal to noise ratio (SNR).
Stellar targets are specified by a Kepler ID, which is used to look up pertinent data in the Kepler Input

Catalog (KIC).[10] Stellar targets may be either LC or SC and are associated with specific assigned pixels via *target definitions*.[8]

Custom targets are explicitly specified collections of pixels. Custom targets are defined by a reference pixel position and a set of offsets, one for each pixel, from that reference position. Custom targets are used for non-stellar sources and diagnostic collections of pixels, and may be either SC or LC.

Background targets are small (nominally 2×2) sets of pixels that sample the background signal in long cadence. These pixels are selected to support a 2D polynomial representation of the background.[8]

Reference pixel (RP) targets are special stellar targets used for diagnostics whose pixels are downlinked bi-weekly via low-bandwidth X-band communications.[11]

1.3 The *Kepler* Data Path

Kepler CCD pixel data is converted into a 14-bit digital signal, which is co-added into 23-bit data (embedded in 32-bit words). A nominal long exposure performs 270 co-adds of the 14-bit data. This data can then have several compression options applied. The following describes the nominal path taken by data on the spacecraft:

- Pixel values are requantized into 16 bits via a pre-defined non-linear lookup table that is designed so that the table step sizes are one quarter of the Poisson shot noise that would be associated with the pixel value.
- Every 48 long cadences an uncompressed baseline is stored to support the Huffman encoding in the next section. The difference between this and the previous baseline is taken, and the Huffman encoded differences are also stored to protect against data loss.
- The non-baseline cadence pixel values are subtracted from the latest baseline, and the differences are compressed via Huffman encoding.
- The resulting data are packed into Consultative Committee on Space Data Systems (CCSDS) data source packets and stored in the *Kepler* solid-state recorder.
- For monthly downlink the pixel data are packed first into Virtual Channel Data Units (VCDUs), which are then packaged into Channel Access Data Units (CADUs) with Reed-Solomon encoding, randomization and convolution encoding to increase robustness against data loss during transmission.
- The CADU data is transmitted to the the NASA Deep Space Network (DSN), which unpacks the data into VCDUs which are delivered to the *Kepler* Mission operations center (MOC).
- The MOC further unpacks the data and delivers it to the Data Management Center (DMC) who extracts the actual pixel data values and delivers it as FITS files to the *Kepler* Science Operations Center (SOC) for processing.
- The SOC Data Analysis Pipeline processes the data, calibrating the pixels, performing photometry, searching for planetary transits and other astrophysical phenomena and monitoring spacecraft health.

The SOC data analysis pipeline includes the following steps:

Pixel-level calibration[7] which removes the CCD bias and dark levels, applies a flat field and removes smear due to shutterless operation.

Creation of flux light curves[22] via simple aperture photometry, including background processing and cosmic ray removal.

Pre-search data conditioning[7] which removes various systematics from the flux light curves.

Transiting planet search[7] which whitens the flux time series for each target and applies the transit search algorithm.

Data validation[?] which applies various tests to provide confidence metrics for the results of the transiting planets search.

ETEM is tasked with generating high-fidelity synthetic data that exercises every step in the above chain with realistic simulations of *Kepler* pixel data containing expected astrophysical effects including planetary transits and spacecraft noise and systematics. In the above chain of data, the simulated pixel value up to requantization into 16 bits is implemented in MATLAB, and the Huffman encoding and packaging of the data is implemented in java. ETEM is designed to be extensible in several ways, allowing increased knowledge of the spacecraft and astrophysics to be inserted as needed.

2. EFFICIENTLY SIMULATING ASTROPHYSICAL PHENOMENA

An ETEM simulation is performed for a single output channel at a time, and takes as input a variety of data:

The *Kepler* Input Catalog[10] (KIC) providing information about stars in the *Kepler* field.

The Pixel Response Function[20] (PRF), an observation-based super-resolution representation of how starlight falls on pixels. The PRF includes the optical point spread function convolved with intra-pixel variability and high-frequency pointing jitter.

Target definitions,[8] that define the target stars and which pixels are to be observed in LC, SC and RP.

Solar-like variability model

Pointing jitter model, an estimate of the low-frequency spacecraft pointing jitter.

Focal plane geometry (FPG) and pointing model,[21] which includes measurements of the locations of the CCDs in the *Kepler* focal plane, models of the *Kepler* optics and of DVA. These models are used to determine the pixel location of the central ray of each stellar target.

Saturation model,[21] which includes information about the well depth of each output channel.

Other data about CCDs and system electronics,[21] such as flat fields, CCD charge diffusion and charge transfer efficiency (CTE), electronic dark levels, observed instrumental noise.

Cosmic rays are simulated as described in the 2004 ETEM paper.[13]

The above data are provided to ETEM by a user-created MATLAB script. This input script specifies various parameters and which classes and plugins implement the above models. Most models have a variety of plugins available, allowing the user to choose what phenomena are simulated in a particular run (see §4).

2.1 Simulating stars, their positions and motions

ETEM simulates dynamic stellar brightness modulations only for stars on the observed target list defined by the input target definitions (§1.2). All other stars are considered to have static magnitudes, and shot noise is injected into all pixels based on their values. All stars are subject to DVA and pointing jitter motions.

A primary concern of ETEM is the ability to produce 90 days of simulated data for all 84 channels in a reasonable amount of time. This is facilitated by parallelizing the simulation, computing each output channel's data on a separate system in the SOC cluster.[3] But a single channel contains tens of thousands of stars in the KIC including about 2000 observational targets that require dynamic modulations. Significant performance speedup is achieved by performing the simulation at the co-added LC or SC cadence time resolution rather than simulating individual 6.5s exposures. LC and SC simulations are performed separately.

Another significant performance-enhancing strategy is based on the insight that stellar motions due to DVA and pointing jitter are critical systematics in *Kepler* observations and therefore must be modeled by ETEM. Rather than rendering all stars on an output channel from scratch, which would be quite slow, ETEM develops a linear polynomial model of the response to each pixel's starlight to motion of that star on a sub-pixel (nominally 0.1 pixel) grid. This approach takes advantage of the highly optimized linear algebra algorithms used by MATLAB. This strategy has been described in a previous paper.[13] Here we provide a brief summary.

First, each star in the KIC that falls on the output channel being simulated is projected onto the sub-pixel grid. Then a polynomial representation $PRF\left(\Delta x, \Delta y\right)$ of the PRF for this channel as a function of offset $(\Delta x, \Delta y)$ is created on the sub-pixel grid that covers the extent of DVA and jitter motion. For each sub-pixel position, the flux (as determined from the KIC) of all stars projected on that position is summed and then convolved with each coefficient of $PRF\left(\Delta x, \Delta y\right)$. The result is a representation of the flux in pixel (r, s) of the form

$$p_{r,s}\left(\Delta x, \Delta y\right) = \sum_{i,j=0}^{O} c_{i,j,r,s} \Delta x^i \Delta y^j. \tag{1}$$

Here the coefficients $c_{i,j,r,s}$ is the convolution of the flux falling on pixel (r, s) with the corresponding coefficient of $PRF\left(\Delta x, \Delta y\right)$, and O is the number of coefficients as determined by the order of the polynomial. This convolution gives a set of polynomial coefficients for all pixels on the output channel pixel array, which can be quickly evaluated for any small offset $(\Delta x, \Delta y)$. The polynomial coefficients for each observed target star are also stored separately to facilitate the addition of dynamic signals to these stars as described below.

The offsets $(\Delta x, \Delta y)$ are themselves represented by a 2D polynomial fit to a grid of artificial "stars" projected on the sky. DVA and motion jitter models are applied to these fake stars and their resulting positions fit with a 2D *motion polynomial* for each simulated cadence. This structure allows an entire channel of flux data to be filled by first evaluating the motion polynomial on all pixels, which provides $(\Delta x, \Delta y)$ for each pixel, and then evaluating (1) on these offsets for each pixel. Polynomial evaluations are very fast in MATLAB: all pixels on the channel are rendered by a small number of polynomial evaluations. An image resulting from the evaluation of these polynomials at a single cadence is shown in Figure **??**.

Observational targets have further brightness modulation due to stellar variability (§2.1.1) and transit models (§2.1.2). For each target, the light curves from these modulations are multiplied together to form the final modulation light curve L_m, which is normalized to 1 when there is no modulation. For each cadence, the stored pixel flux polynomials for each target are evaluated using $(\Delta x, \Delta y)$, producing the pixel values P_{target} for that target only. A copy $P_{\text{modulated}}$ of P_{target}, and the difference $P_{\text{target}} - L_m P_{\text{modulated}}$ is added to the simulated pixel array containing all targets at the target's position. As a result the target's contribution to its pixels is modulated by L_m.

The above polynomial framework is not suited for saturated targets, whose flux spills up and down CCD columns. *Kepler* includes saturated targets in its observations because saturation spill in the *Kepler* CCDs is conservative. Therefore the flux of a saturated target can be measured so long as all saturated pixels are captured. Saturated targets are treated in ETEM as a special case after the polynomial pixel rendering, simulating saturation spill on a target-by-target basis.

2.1.1 Stellar variability

Stellar variability is modeled using a 4-year observation of Solar observations from the DIARAD instrument on the Solar Heliospheric Observatory.[12] Differences in variability among different stars is modeled using the relationship between photometric variability σ_{phot} and rotation period P_{rot} given by[16, 17] $\sigma_{\text{phot}} \approx P_{\text{rot}}^{-1.5}$. Observed target stars are randomly assigned a rotation period within a specified range centered on the Solar rotation period. Frequencies in the DIRAD data are scaled to the chosen rotation period, and the resulting time series is scaled to σ_{phot}. There is no attempt to match these parameters to stellar properties in the KIC, so all observed targets are assigned near-Solar variability regardless of their luminosity class. Because stellar variability is implemented as a plugin (see §**??**) it is easy to implement other stellar variability models.

2.1.2 Transit simulation

User input determines which observed targets are assigned eclipsing binary and/or transiting planet light curves. The user can assign a range of target parameters such as magnitude or surface gravity, in which case a specified number of observed targets are selected from these ranges. Alternatively a specific *Kepler* ID in the KIC can

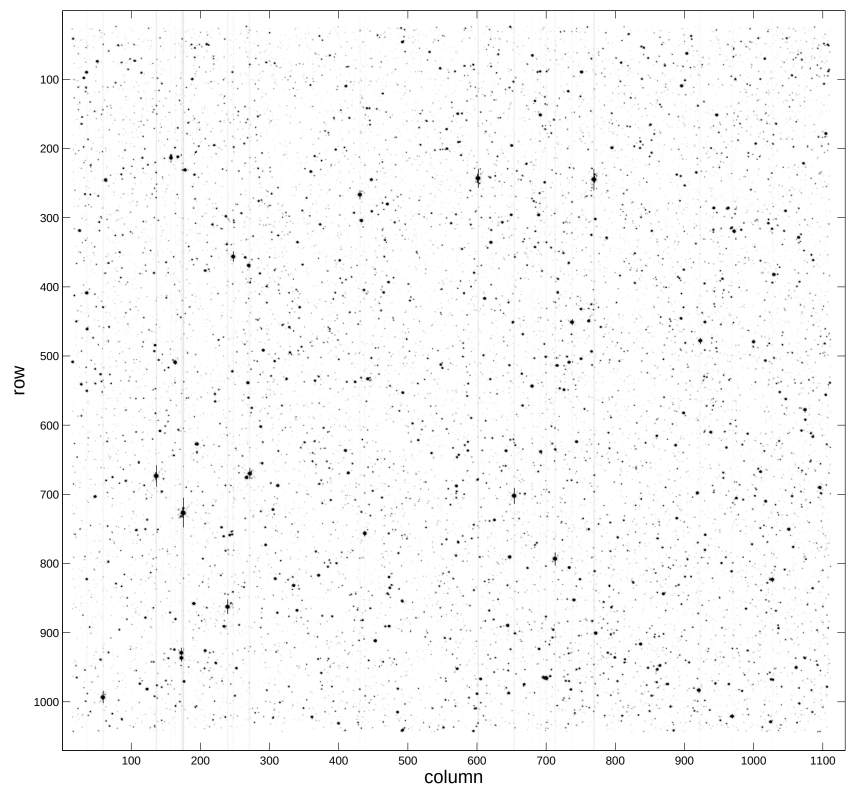


Figure 2. An image of an output channel simulated in ETEM based on the KIC, showing stars, including saturated stars with saturation spill along vertical columns, as well as faint smear trails.

be specified. Parameters of the eclipse/transit are chosen from user-specified ranges. For eclipsing binaries these user-specified ranges include orbital period, minimum impact parameter, secondary star logG and effective temperature while eccentricity is chosen from the model of Duquennoy and Mayor.[18] For planetary transits these parameters include period, minimum impact parameter, eccentricity and radius. The orientation of the orbit relative to the line of sight, which determines the epoch, is randomly chosen in a way that delivers the specified impact paramter. Primary star properties are obtained from the KIC when possible, otherwise they are drawn from the Besancon model.[19] These parameters are used to construct a Keplerian orbit, from which a high-time-resolution time series of the secondary impact parameter is computed. Care is taken to assure that the periastron of an orbit occurs outside the primary star's radius, possibly overriding the input orbital period, but there is no attempt to enforce dynamical constraints when multiple planet systems are defined. It is up to the user to avoid specifying unrealistic planetary systems.

Given an impact parameter time series, eclipsing binary and planetary transit light curves are simulated using the analytic model of Mandel and Agol.[14] This model provides a direct simulation of the transit light curve accounting for nonlinear limb darkening. The MATLAB implementation of this method was based on an IDL script provided by Agol, available at http://www.astro.washington.edu/users/agol/. ETEM uses nonlinear limb darkening coefficients from the Atlas model.[15] For eclipsing binaries the resulting transit signals are scaled according to the relative brightnesses of the stellar components and whether the eclipse is a primary or secondary. A simulated Jupiter-size transit is shown in Figure **??**.

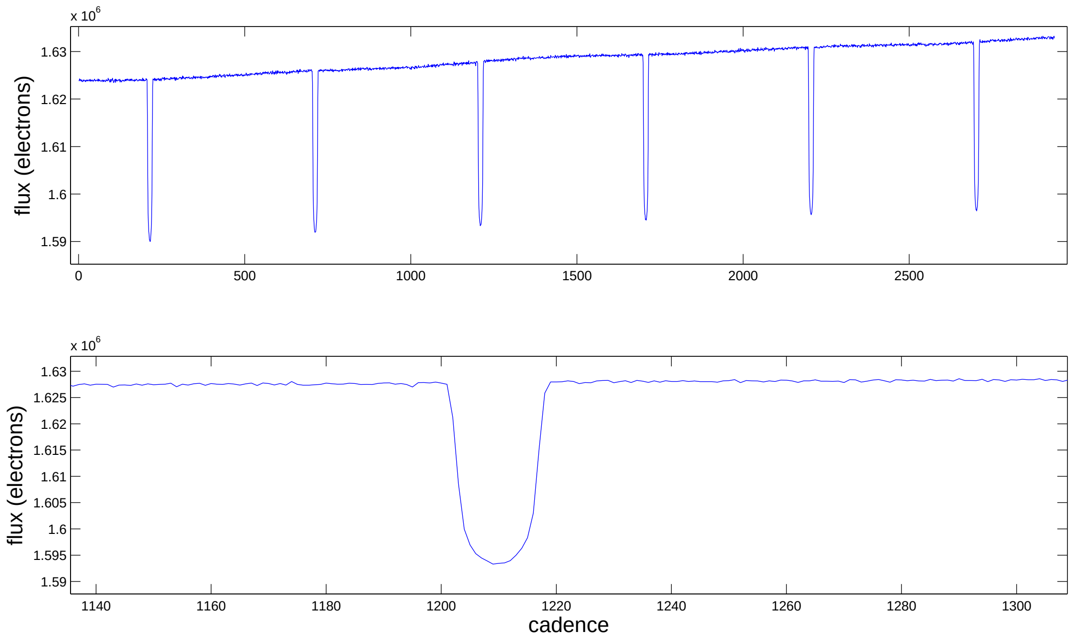


Figure 3. The light curve of a transiting Jupiter-size planet simulated in ETEM, with a closeup in the bottom panel. The effects of limb darkening are visible in the closeup.

3. A FRAMEWORK TO SUPPORT EXTENSIBILITY

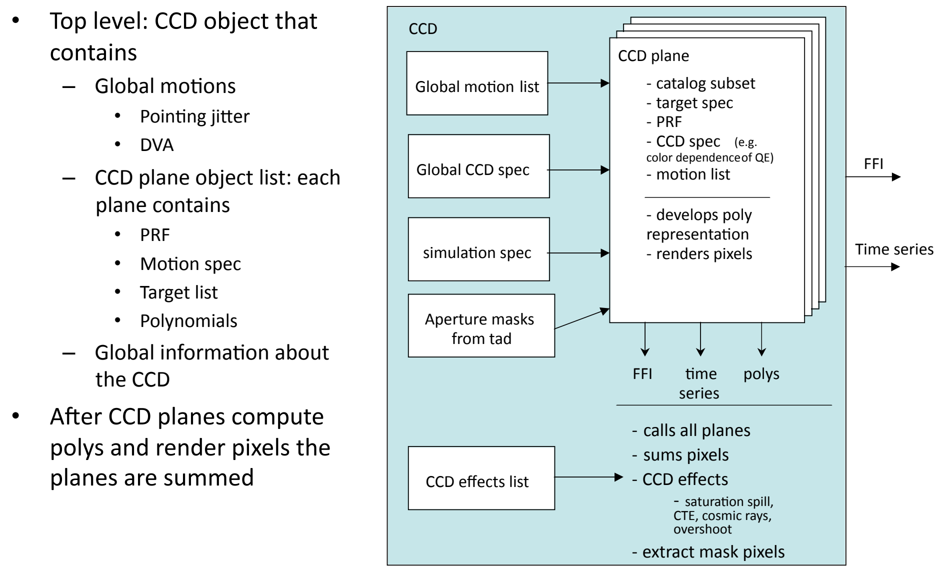
ETEM is designed to support testing both the flow of data and the the SOC data analysis pipeline from pixel calibration through planet detection described in 1.3. This entails simulating many aspects of the data collected by the spacecraft, and how that data is flowed though the *Kepler* system. ETEM is required to respond to new and unexpected information that has become available throughout the multi-year *Kepler* mission, allowing contributions from multiple developers. These developers must not be required to understand the full ETEM system in order to contribute new functionality.

The simulated aspects fall into two groups:

Simulations of astrophysical phenomena and spacecraft properties and systematics, which are subject to change as our knowledge of the spacecraft and sky improve in flight. Different phenomena are included in a particular simulation run in order to isolate the response of the system to individual phenomena, reflect improved knowledge of the sky or spacecraft, and allow controlled tests of non-realistic inputs.

Simulations of data formats at various stages of the flow of data through *Kepler* ground processing. These formats are rigidly fixed and so is not dynamic

Because the desired astrophysical phenomena or spacecraft properties in a simulation can change over time, an extensible object-oriented framework that implements a plug-in philosophy was chosen. The plugin philosophy is chosen to simplify the addition of new pheneomena to ETEM, satisfying the requirement that developers need not be expert in all of ETEM in order to add functionality. This framework is implemented in MATLAB, while the relatively rigid simulation of formats is implemented in java. This rest of this paper describes the MATLAB implementation of the simulation framework.



- Top level: CCD object that contains
  - Global motions
    - Pointing jitter
    - DVA
  - CCD plane object list: each plane contains
    - PRF
    - Motion spec
    - Target list
    - Polynomials
  - Global information about the CCD
- After CCD planes compute polys and render pixels the planes are summed

Figure 4. The ETEM class structure. The ccd class contains a list of several global classes including noise and motion models, as well as a list of ccdPlane classes. Each ccdPlane class contains target lists, the PRF class and motions for targets on that ccdPlane.

3.1 The ETEM Class Structure

ETEM is implemented in a version of MATLAB that did not support full object-oriented class hierarchies. Therefore the class design in ETEM is relatively flat, with a few high-level classes managing lists of lower-level classes. Class inheritance and subclassing is used only sparingly. If ETEM were to be implemented in more recent versions of MATLAB there would be much wider use of inheritance and subclassing.

All classes in ETEM are sub-classes of the *runParams* class, which contains the specification of all parameters and plugins in a simulation. The runParams class is instantiated using data from a user-defined input script that specifies all parameters of an ETEM simulation. The runParams class contains *get* but no *set* methods, so other classes cannot change the runParams class once it has been created. In this way ETEM enforces data consistency and ensures that objects are well-encapsulated from each other while providing global access to the simulation's specification.

The top-level class in ETEM is the *ccd* class (Figure **??**) which contains all objects instantiated in an ETEM run. The ccd class represents an output channel and is responsible for computing and rendering simulated pixel values. The ccd class includes a list of *ccdPlane* objects, as well as global motion plugins that are applied to all ccdPlanes. The pixel computation method described in §2 is implemented in the ccdPlane class, which contains a list of stars rendered by that ccdPlane object, and slots for PRF and motion plugins. This structure allows, for example, the simulation of pointing jitter and DVA motion for all stars on an output channel with parallax motion on a small number of stars by having most stars in one ccdPlane object and the parallax stars in a second ccdPlane object. In this example the pointing jitter and DVA motion are in the ccd object's global motion list while the parallax motion is in the second ccdPlane object's motion list. Similarly intra-channel focus variation can be simulated by using two ccdPlane objects with with differing PRFs.

The ccd object also includes plugins controlling global CCD properties such as read noise, well depth and saturation spill symmetry.

The ccd object renders stellar pixel images in two ways in a nominal ETEM run: a rendering of all pixels on the channel creating an FFI for the simulated channel at a specified time, and a rendering of *flux time series* of target pixels for all cadences in a simulation, which includes the modulation of target pixel fluxes over time. In both cases the ccd object calls the render method of the appropriate type (FFI or flux time series) in each ccdPlane object in the ccd object's list of ccdPlanes. Background and other global flux signals are rendered in the first ccdPlane's pixels. The resulting pixels sets, one for each ccdPlane, are summed by the ccd object to produce the final rendered pixels.

The light curves used to modulate a target star are created by *lightcurve* classes, with a class for each type such as a sohoStellarVariability, transitingPlanet or transitingStar class. The transiting object classes contain a transitingOrbit class which collects the methods and data structures that are common to all transiting objects. An example of the difference between the transiting planet and the transiting star classes include accounting for flux from both stars in the transiting star class. The light curve for each of these classes is computed immediately after the class is instantiated and stored in the resulting object.

Astrophysical signals on ETEM simulation targets are managed by the *targetScienceManagement* class. This class contains a list of *targetSpecification* structures, one for each target. Each target's list entry contains a list of data structures defining what lightcurve classes are associated with that target as well as the objects instantiated from those data structures. For example, a target's list may contain an object implementing a stellar variability model in the first entry, and an object for modeling a transiting planet in the second entry. The light curve computation method for each lightcurve object of this list is called, producing a light curve normalized to [0,1] for each list entry. These light curves are multiplied together to produce the final light curve for this target, which is used in the rendering of the target pixels in the ccdPlane object containing this target.

The differences between long and short cadence simulations is managed in two ways: the input parameters determine the cadence time (nominally about 30 minutes for long cadence, about one minute for short cadence), and the differing output data is handled by instantiating either the *longCadenceData* or *shortCadenceData* class. These classes have methods that convert the rendered pixel flux time series into the format appropriate to the cadence type. The resulting object is contained in the ccd class, and the data formatting methods are called after each cadence of pixels is computed.

3.2 ETEM Plugins

An ETEM plugin is a class, defined as a particular plugin by its interface. For example the read noise plugin has two methods: a creation method and an *apply_noise* method that takes as input an output channel's pixels and returns the pixels with the specified noise added. Plugins with differing interfaces are said to have different *plugin types*. Each plugin type has a different slot in either the ccd or ccdPlane class.

Several plugin types are currently supported in ETEM, with the class that contains the plugin given in parenthesis followed by different examples of each plugin:

PRF (ccdPlane class): pre-flight simulated or flight-measured PRFs.

TAD input (ccd class): flight target definitions or target definitions created from scratch according to some specification.

Catalog reader (ccd class): stellar catalog sources. Implemented plugins include the KIC and custom star catalogs defining artificial star fields (such as a grid of equally-spaced equal-magnitude stars) for controlled tests.

Star selector (ccdPlane class): algorithmic selection of stars on a ccdPlane. Examples include stars selected by position or magnitude, or by explicit specification via *Kepler* ID number.

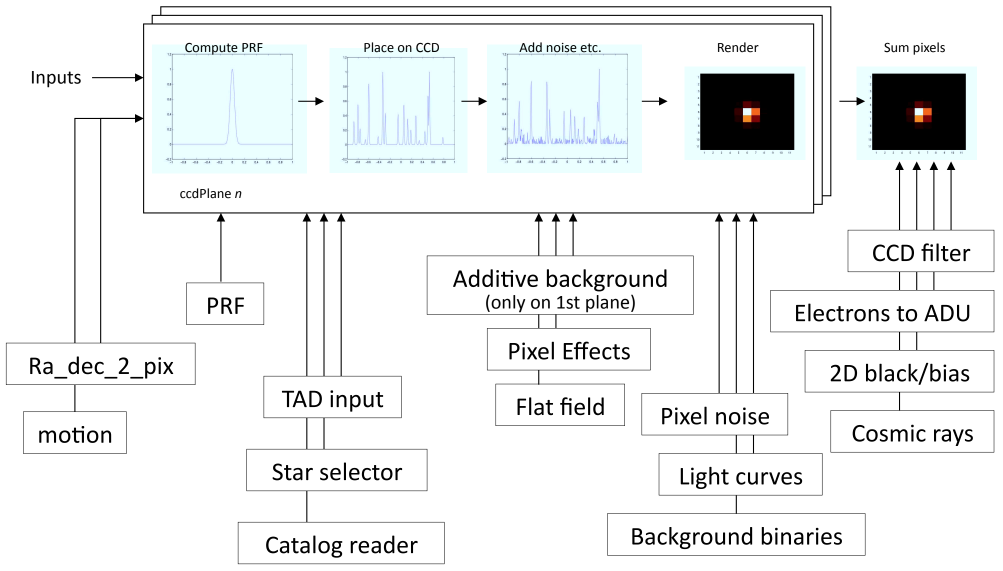


Figure 5. The ETEM plugin structure, showing how the plugins relate to the rendering of pixel values. Some of these plugins are in the ccd object and are applied to all ccdPlanes. Other plugins are specific to particular ccdPlanes.

Additive background (ccd class): additive backgrounds such as zodiacal light or diffuse stellar background.

Cosmic ray (ccd class):models to add cosmic rays to rendered pixels.

Linearity (ccd class): specifies the linearity function, required to model near-saturation behavior.

Motion (ccd and ccdPlane classs): motion inputs such as pointing jitter or DVA motion to the computation of the $(\Delta x, \Delta y)$ used on §3.2

Flat-field (ccd class): flat fields, such as inter-pixel variability and vingetting, to apply to the pixel values.

Well-depth (ccd class): well-depth functions, including a single scalar value or spatially-varying values.

Charge transfer efficiency (ccd class): the function that defines charge transfer efficiency.

Gain modulation (ccd class): time or position dependent pixel gain.

Bias (ccd class): various bias functions added to the pixel values, including time, space and temperature dependence.

Pixel noise (ccd class): pixel-value-dependent noise, such as shot noise.

Read noise (ccd class): output-channel-dependent read noise, which may be scalar or spatially dependent.

Cosmic ray (ccdPlane class): specify the simulation of cosmic rays.

Barycentric correction (ccdPlane class): specify time offset functions for light curve generation.

Light curve (ccdPlane class): light curve generation such as stellar variability and planetary transits.

The relationship of these plugins to the computation of pixel values is shown schematically in Figure **??**

4. TESTING EXPERIENCE

The use of ETEM to generate simulated data has been absolutely crucial in pre-flight testing of the *Kepler* ground system. Several problems in the flow of the data between organizations were uncovered and solved using ETEM data. The SOC processing pipeline was debugged and shown to successfully identify transits using this data.

The primary motivation for the software structure described in this paper is to enable new functionality to be added to ETEM in a well-encapsulated way. This was tested by the addition of 30 new capabilities after the basic framework was in place. Some of these capabilities were programmed by developers who had essentially no knowledge of the inner workings of ETEM.

The ability of ETEM to simulate new, unexpected phenomena was demonstrated when, in the last year before launch, unexpected time and temperature dependent bias signals were discovered in the *Kepler* pixel values. These signals appear as an additive bias, and such a dynamic bias was not in the ETEM baseline design. It was straightforward to create new plugins that simulated the new signal by simply extending the existing bias plugin, adding the cadence time to its interface. Resulting simulations were used to show that the newly discovered signal did not compromise mission-critical commissioning activities. This new plugin also allows the ongoing assessment of SOC processing pipeline methods of correcting for this bias.

ETEM continues to be used to verify SOC processing software and test modifications to the SOC processing pipeline. Occasionally these tests require new plugins to be defined, and (so far) the ETEM structure has provided the desired support for such testing with a small amount of effort. There is little doubt that the time invested in creating the extensible software structures described in this paper has saved significant time and effort during testing.

5. CONCLUSIONS

This paper describes a robust extensible framework for simulating *Kepler* data. This framework has been used extensively in the *Kepler* mission, both for pre-flight testing of the *Kepler* ground processing system and the result of data processing, specifically the identification of planetary transits. The robust extensible nature of the ETEM framework allows simulation of new phenomena discovered in flight, as well as the testing of new algorithms developed to respond to these phenomena. The ability of developers new to ETEM to contribute new functionality without knowledge of the inner workings of ETEM has been demonstrated on several occasions.

ACKNOWLEDGMENTS

The authors would like to thank the entire *Kepler* team for their support in the development of ETEM. The *Kepler* science office, particularly science office directors Janice Voss and Michael Haas, has been crucial in its support and guidance. We thank *Kepler* management, particularly SOC manager Dave Pletcher and *Kepler* ground segment system engineer Rick Thompson, who gave firm guidance when needed and stayed out of our way the rest of the time. We can't thank the *Kepler* SOC personnel enough for their heartfelt and cheerful support even when ETEM was obviously doing the wrong thing. Finally we thank Ed Gamble for helping to create the initial extensible ETEM concept and sell it to *Kepler* management.

Funding for this Discovery mission is provided by NASA's Science Mission Directorate.

REFERENCES

[1] Borucki, W. *et. al.*, “Kepler planet-detection mission: introduction and first results”, *Science*, **327**, pp. 977–980, (2010).

[2] Koch, D. G. *et. al.*, “Kepler mission design, realized photometric performance, and early science,” *ApJL.*,**713(2)**, pp. L79–L86, (2010).

[3] Middour, C. K. *et. al.*, “Kepler Science Operations Center Architecture,” SPIE 7740, (2010).

[4] Haas, M. *et. al.*, “Kepler Science Operations,” *ApJL.*,**713(2)** L115-L119, (2010).

[5] Jenkins, J. M. *et. al.* “An Efficient End-To-End Model for the Kepler Photometer,” SPIE 5497, (2004).

[6] Van Cleve, J. and Caldwell, D. A., Kepler Instrument Handbook, KSCI 19033-001, (Moffett Field, CA: NASA Ames Research Center) (2009).

[7] Caldwell, D. A. *et. al.*, “Instrument Performance in Kepler’s First Months,” *ApJL.*,**713(2)**, L92-L96. (2010).

[8] Quintana, E. V. *et. al.*, “Pixel-level calibration in the Kepler Science Operations Center pipeline,” SPIE 7740, (2010).

[9] Bryson, S. T. *et. al.*, “Selecting Pixels for Kepler Downlink,” SPIE 7740, (2010).

[10] Batalha, N. M. *et. al.*, “Selection, Prioritization, and Characteristics of Kepler Target Stars,” *ApJL.*,**713(2)**, pp. L109–L114, (2010).

[11] Latham, D. W. *et. al.*, “The Kepler input catalog,” Proc. AAS 207, 1340 (2005).

[12] Chandrasekaran, H. *et. al.* “Semi-weekly monitoring of the performance and attitude of Kepler using a sparse set of targets,” SPIE 7740, (2010).

[13] Twicken, J. D. *et. al.*, “Photometric Analysis in the Kepler Science Operations Center Pipeline,” SPIE 7740, (2010).

[14] Twicken, J. D. *et. al.* “Presearch data conditioning in the Kepler Science Operations Center pipeline,” SPIE 7740, (2010).

[15] Jenkins, J. M. *et. al.* “Transiting planet search in the Kepler pipeline,” SPIE 7740, (2010).

[16] Wu, H. *et. al.* “Data Validation in the Kepler Science Operations Center Pipeline,” SPIE 7740, (2010).

[17] Bryson, S. T. *et. al.* “The Kepler Pixel Response Function,” *ApJL.*,**713(2)**, pp. L97–L102, (2010).

[18] Allen, C. A. *et. al.*, “Kepler mission’s focal plane characterization models implementation,” SPIE 7740, (2010).

[19] Jenkins, J. M. “The Impact of Stellar Variability on the Detectability of Transiting Terrestrial Planets,” *ApJ*, **575**, pp. 493-505, (2002).

[20] Noyes, R. W. *et. al.*, *ApJ*, **279**, 763, (1984).

[21] Radick, R. R. *et. al.*, *ApJS*, **118**, 239, (1998).

[22] Duquennoy, A. and Mayor, M. *A&A*, **248**, 485, (1991).

[23] Robin, A. C. *et. al.* “A synthetic view on structure and evolution of the Milky Way,” *A&A*, **409**, 523, (2003).

[24] Mandel, K. and Agol, E. “Analytic Lightcurves for Planetary Transit Searches,” *A&A*, **363**, 108C, (2000).

[25] Claret, A. “Non-linear limb-darkening law for LTE models,” *ApJ*, **580**, pp. L171-L175, (2002).